Rings and Modules in Isabelle/HOL

Hidetsune Kobayashi, Hideo Suzuki, Hirokazu Murao

Formalization of Rings, Modules over a ring in Isabelle/HOL.

Why Isabelle/HOL?

We can formalize abstract ring theory.

Why abstract theory?

Use the abstract ring theory for studying the multiplicity of a solution to a system of algebraic equations.

Why not symbolic computation?

We want to calculate not only formulae but also abstract theory.

What we have done, concerning the multiplicity

1. Construction of a U-resultant by using Gröbner basis.

   The multiplicity is equal to the multiplicity of an algebraic equation.

   ..... symbolic computation and numerical calculation.

2. Calculation of the multiplicity by using an extended Zeuthen's rule.

   In almost any case, we can calculate an upperbound and a lower bound of the multiplicity.

   ..... symbolic and numerical calculation.

Where to go?

Formalize multiplicity theory of local rings. It includes dimension theory, spectral sequence, derived functors, etc.

Texts are

1. M. F. Atiyah, I. G. Macdonald, “Introduction to Commutative Algebra”.

2. J. P. Serre , “Algèbre Locale, Multiplicités”

Now, chapter 1, 2 of [1], set theory and group theory are formalized.

What we talk

1. a short introduction to Isabelle/HOL

2. report on some examples of formalization
   - 2-1. modules over a ring
   - 2-2. generators of a module
   - 2-3. finitely generated modules
   - 2-4. Nakayama lemma
   - 2-5. tensor products

3. some examples we cannot formalize well

A short introduction to Isabelle/HOL

example is a formalization of an ordered set.

```
record 'a OrderedSet =                    (* create a record for an ordered set *)
 base_set  :: "'a set"
 ord_rel  :: "('a  * 'a) set"
 ordering  :: "['a, 'a] => bool"

constdefs
 (* definition of the ordered set *)
 ordered_set :: "'a OrderedSet => bool"
   "ordered_set D == (ord_rel D) ⊆ (base_set D) × (base_set D) ∧ ... "
 (* strict order *)
 ord_neq :: "['a OrderedSet, 'a, 'a] => bool"
   "ord_neq D a b == ordering D a b  ∧  a ≠ b"
```

constdefs

```
 Iod :: "['a OrderedSet, 'a set] => 'a OrderedSet"
  "Iod D T == (| base_set = T, ord_rel = {x. x ∈ ord_rel D ∧ (fst x) ∈ T ∧
                      (snd x) ∈ T}, ordering = ordering D |)"
```

(* T is a subset of an ordered set D. Iod is an ordered set with base_set T *)

syntax

```
 "@ORDERING"::"['a, 'a OrderedSet, 'a] => bool"
   ("(3/ _/ '≤_/ _)" [100,100,101]100)

 "@ORDNEQ"::"['a, 'a OrderedSet, 'a] => bool"
   ("(3/ _/ '<_/ _)" [100,100,101]100)
```

translations

"a $<_D$ b" == "ord_neq D a b"

"a $\leq_D$ b" == "ordering D a b"

By the translation above, we can write a $<_D$ b instead of ord_neq D a b.

R modules

Module is defined as the ordered set. To discuss the exact sequence of modules of homomorphisms, we need a Module whose carrier is the set of module homomorphisms.

constdefs
HOM :: "[('r, 'more) ringtype_scheme, ('a, 'r, 'more1) moduletype_scheme,
('c, 'r, 'more1) moduletype_scheme] => ('a => 'c, 'r) moduletype"
("(3HOM_/ _/ _)" [90, 90, 91]90)

"HOM_R M N == (| carrier = mHom R M N, abOp1 = bOp1_mHom R M N,
aiOp1 = iOp1_mHom R M N, aunit1 = mzeromap M N, sprod =sprod_mHom
R M N |)"

Here, mHom R M N is the set of module homomorphisms of M to N. Operators are defined as operations of functions. We gave a formalized proof that HOM_R M N is an R-module.

Chinese remainder theorem

theorem Chinese_remThm:"[| Ring R;
 (∀k ∈ Nset (Suc n). ideal R (J k)); (*∀k, (J k) is an ideal of R, $0 \leq k \leq n + 1$*)
 ∀k ∈ Nset (Suc n). B k = QRing R (J k);
 (*∀k, (B k) is equal to $R /_r$ (J k), $0 \leq k \leq n + 1$ *)
 ∀i ∈ Nset (Suc n). ∀j ∈ Nset (Suc n). (i ≠ j --> coprime_ideals R (J i) (J j))
 (* (J i) and (J j) are coprime ideals *)
|] ==>
 $R /_r (\cap \{J\ k \mid k.\ k \in Nset\ (Suc\ n)\})) \cong_R (r\prod_{(Nset\ (Suc\ n))} B)$ "

This expression is complicated a little, but comments will help you to see this expression is equivalent to

$$R /\cap (J\ k) \cong_R (\prod R/(J\ k)) \quad (0 \leq k \leq n+1)$$

The isomorphism is well known.

Generator of a Module

constdefs
generator ::"[('r, 'm) ringtype_scheme, ('a, 'r, 'm1) moduletype_scheme,
'a set] => bool"
"generator R M H == H $\subseteq$ carrier M $\wedge$
linear_span R M (carrier R) H = carrier M"

This formalization is quite simple. We defined linear span as

constdefs
linear_span::"[('r, 'm) ringtype_scheme, ('a, 'r, 'm1) moduletype_scheme,
'r set, 'a set] => 'a set"
"linear_span R M A H == if H = { } then $\{0_M\}$ else {x. ∃n. ∃f ∈ Nset n -> H.
∃s ∈ Nset n -> A. x = linear_combination R M n s f}"

Here, A is an ideal of the ring R, so we can treat linear span with coefficients in the ideal A. This enables us to formalize Nakayama lemma.

Finite Generators

If the number of generators is a finite number, we have to sum up coefficients of similar terms.

```
lemma finite_lin_span:"[| Ring R; R Module M; ideal R A;
 h ∈ Nset n -> carrier M; s ∈ Nset na -> A; f ∈ Nset na -> h ` Nset n |] ==>
 ∃t ∈ Nset n -> A.
        linear_combination R M na s f = linear_combination R M n t h"
```

Linear_combination R M n t h stands for $\sum_{i=0}^{n} t(i)\, h(i)$.

This lemma implies if the image of h is a generator of M, then linear combination of any length can be expressed as a linear combination of length n.

Nakayama lemma

lemma NAK:"[| Ring R; R Module M; M fgover R; ideal R A; A $\subseteq$ J_rad R;
A $\odot_R$ M = carrier M |] ==> carrier M = {0_M}"

Here, M fgover R means M is a finitely generated module over R. J_rad R is the Jacobson radical of R. A $\odot_R$ M means the linear span with coefficients in the ideal A of R.

There are two ways of proof, one is using a determinant trick and another is using a decrease in the number of elements of a generator(see[1]).
We formalized the latter.

Nakayama lemma'

Nakayama lemma of quotient module version is

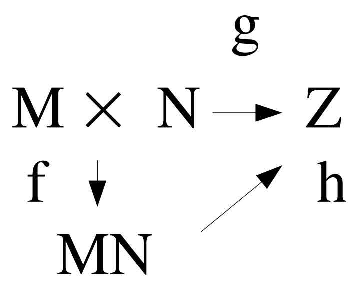
lemma NAK1:"[| Ring R; R Module M; M fgover R; Submodule R M N;
ideal R A; A ⊆ J_rad R; carrier M = A $\odot_R$ M $+_M$ N |] ==> carrier M = N"

Proof of this lemma is easy to formalize.

Tensor Products

Universal property is formalized as follows. A dummy makes it ugly, but we don't know a clean formalization.

```
constdefs
 universal_property::"[('r, 'm) ringtype_scheme,      (* type for R *)
  ('d, 'r, 'm1) moduletype_scheme,   (* type for dummy MV, MV and Z have
                                        the same type *)
  ('a, 'r, 'm1) moduletype_scheme,   (* type for M *)
  ('b, 'r, 'm1) moduletype_scheme,   (* type for N *)
  ('c, 'r, 'm1) moduletype_scheme,   (* type for MN *)
  'a * 'b =>'c] => bool"
 "universal_property R (MV:: ('d, 'r, 'm1) moduletype_scheme) M N MN f ==
  (bilinear_map R M N MN f) ∧
  ( ∀(Z :: ('d, 'r, 'm1) moduletype_scheme). ∀g. (R Module Z) ∧
  (bilinear_map R M N Z g) -->  ((∃!h. (h ∈ mHom R MN Z) ∧
  (compose (M ×_c N) h f = g))))"
```



Why dummy?

Because of type inference, matching fails if we don't assign a type to be matched. And in case of the universal property, the type assigned to Z (appearing as “for all Z”) does not appear on the left hand side if the dummy (having the same type of Z) is not residing, and Isabelle/HOL wouldn't work. This is why we put the dummy.

Existence of the tensor product

constdefs

tensor_product::"[('r, 'm) ringtype_scheme, ('a, 'r, 'm1) moduletype_scheme, ('b, 'r, 'm1) moduletype_scheme] => (('a * 'b => 'r) set, 'r) moduletype"

"tensor_product R M N == $(FM_R\ (M \times_c N))\ /_m\ (TR_R\ M\ N)$"

Here, $FM_R\ (M \times_c N)$ is a module with carrier

{f. f ∈ carrier (M) × carrier (N) --> carrier R ∧ f (x) = 0_R except a finite number of elements x ∈ (carrier (M) × carrier (N))}

and TR_R M N is the submodule generated by the tensor relations.

An example we cannot formalize well

Exact sequence.

$$M_0 \xrightarrow{f_0} M_1 \xrightarrow{f_1} \ldots \rightarrow M_n \xrightarrow{f_n} \ldots$$

We want to assign ('a_i, 'r) moduletype to M_i, because in an exact sequence we have two types of modules, say ('a, 'r) moduletype and ('a set, 'r) moduletype.

Using a generic type gn which matches any type m i, we want to define as

```
constdefs
 exact_sequence::"[('a, 'more) ringtype_scheme, nat,
                  nat => ('gn, 'r) moduletype,  nat => ('gn => 'gn)] => bool"
  "exact_sequence R n M f == Ring R ∧
   ∀j ∈ Nset n. R Module  ((M j)::(m j, 'r)  moduletype)  ∧
   ∀j ∈ Nset n. (f  j) ∈ mHom R  ((M j)::(m j, 'r) moduletype)  ((M (j + 1))::
    (m (j + 1), 'r) moduletype). ... "
```

It seems it is impossible to define like this.

Now, we define exact sequence case by case, i.e. exact3, exact4, ... :

```
constdefs
 exact3 ::"[('r, 'm) ringtype_scheme,
      ('a, 'r, 'm1) moduletype_scheme,   ('b, 'r, 'm1) moduletype_scheme,
      ('c, 'r, 'm1) moduletype_scheme,  'a => 'b, 'b => 'c] => bool"
```

"exact3 R L_0 L_1 L_2 h_0 h_1 == h_0 ` (carrier L_0) = ker $_{L1,\, L2}$ h_1"

(*
$$L_0 \xrightarrow{h_0} L_1 \xrightarrow{h_1} L_2$$
*)

```
 exact4 ::"[('r, 'm) ringtype_scheme, ...              "
```

Hope someone will help us to make a clean formalization.

We cannot write a proof

Proposition

$L_1 \to L_2 \to L_3 \to 0$ (exact) $\Leftrightarrow$

$\forall N.\ \mathrm{Hom}(L_1, N) \leftarrow \mathrm{Hom}(L_2, N) \leftarrow \mathrm{Hom}(L3, N) \leftarrow 0$

(exact)

Because of the type inference of Isabelle/HOL, matching fails when we assign a special module to N. Prof. Ballarin gave us a suggestion, and the problem will be resolved (We hope).

Thank you.